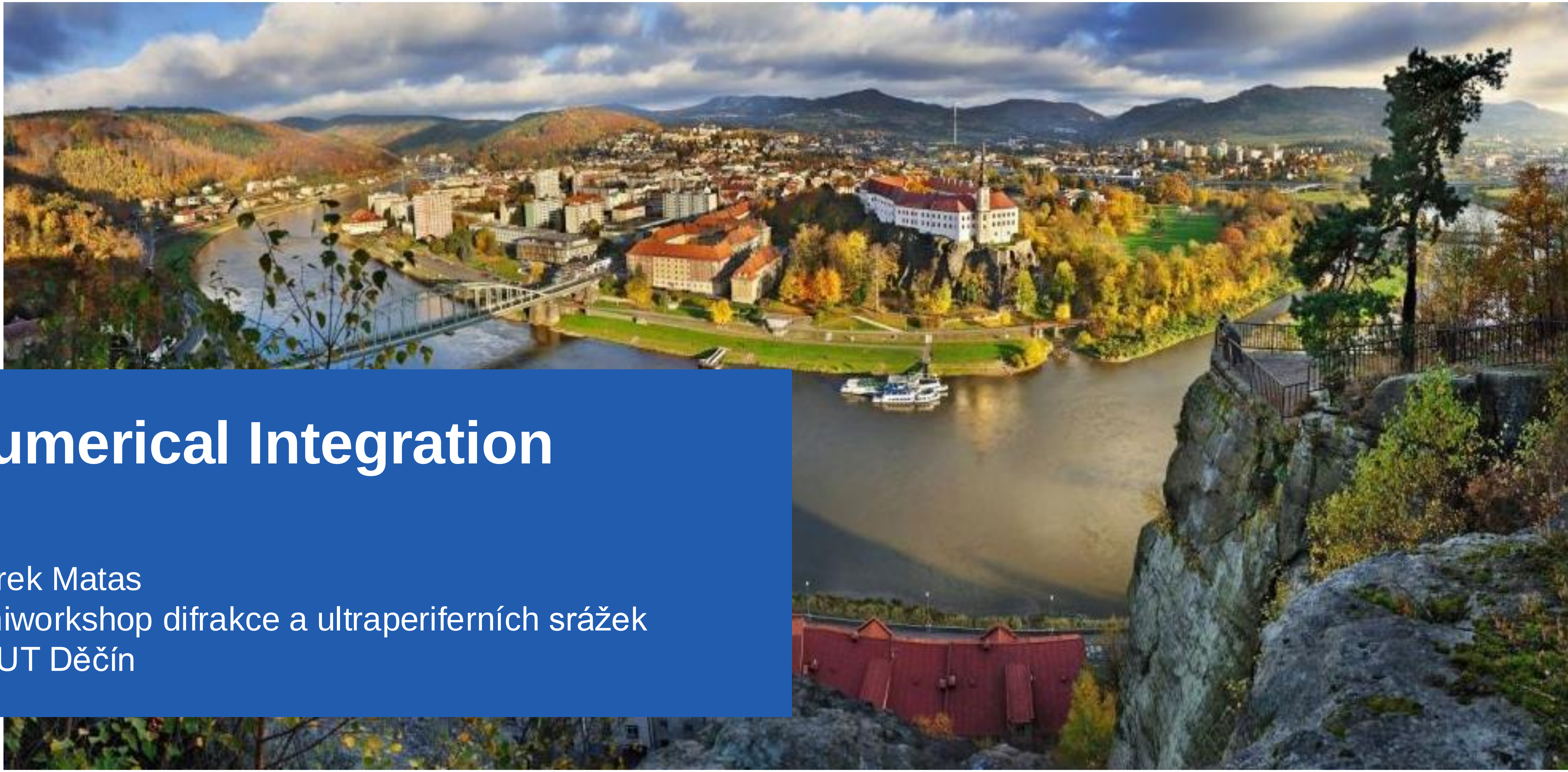


# Numerical Integration

Marek Matas
Miniworkshop difrakce a ultraperiferních srážek
ČVUT Děčín

# HOW DO YOU INTEGRATE ?

# NUMERICAL INTEGRATION

Oldies but goldies

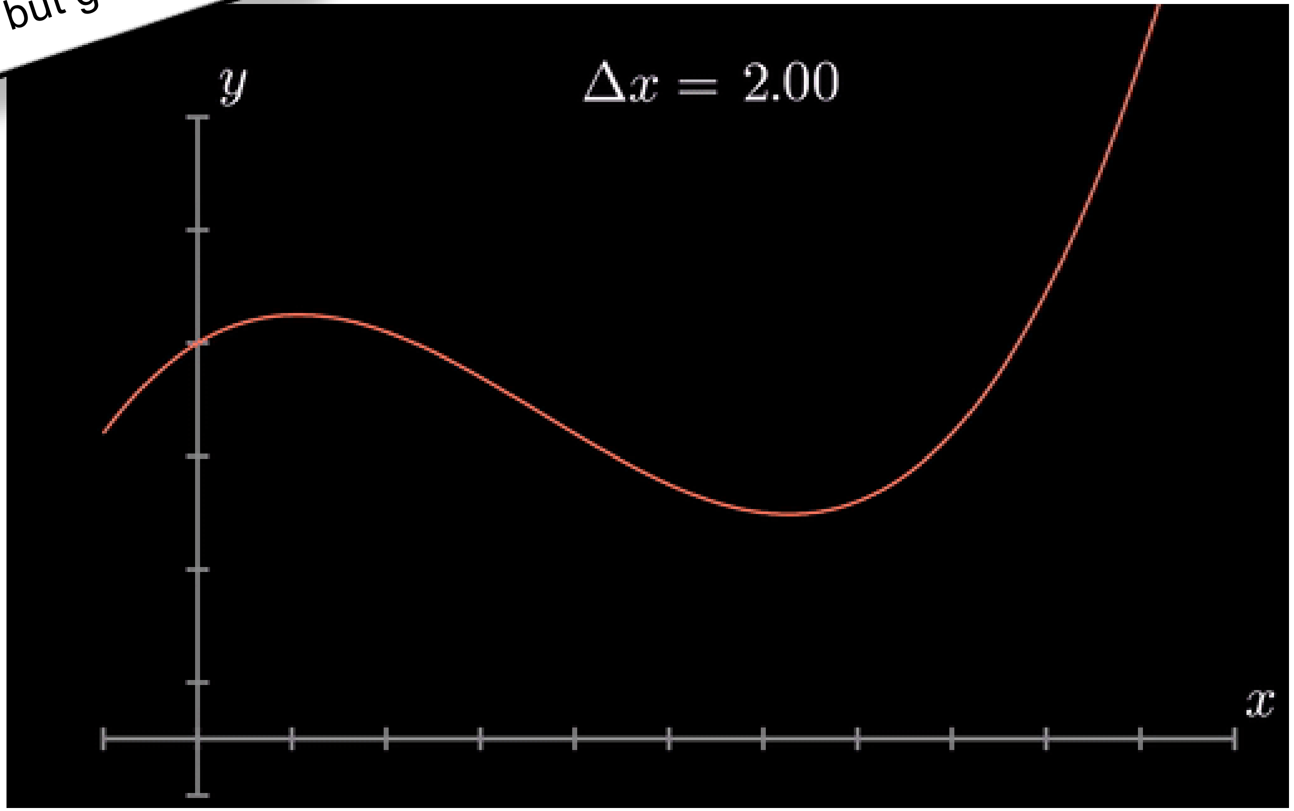
$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$
$$= \frac{1}{3}h\left[f(a) + 4f\left(a+h\right) + f(b)\right],$$

# THE SIMPSON METHOD

# NUMERICAL INTEGRATION

Oldies but goldies

# THE SIMPSON METHOD



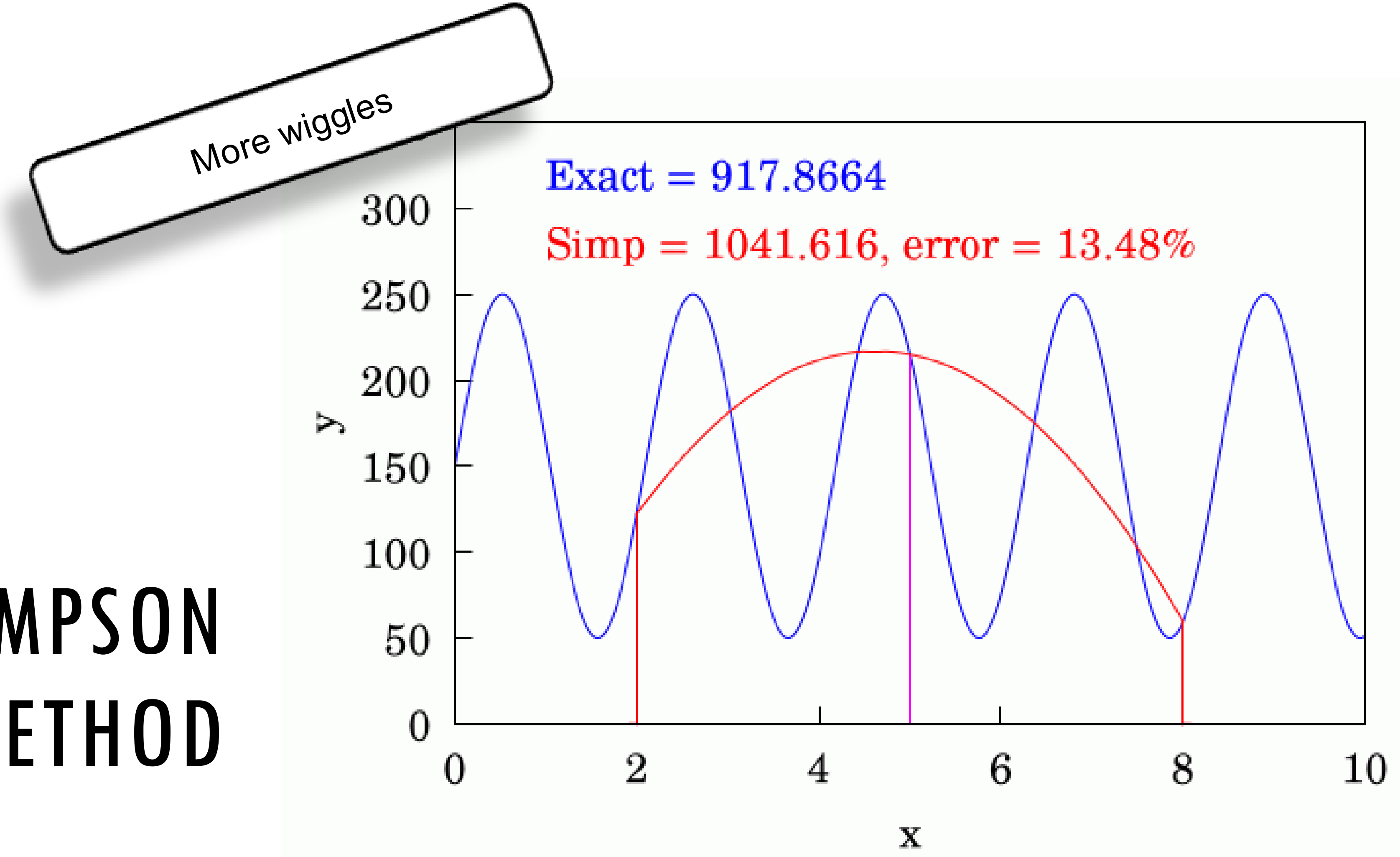
# NUMERICAL INTEGRATION

More wiggles

# THE SIMPSON METHOD

# NUMERICAL INTEGRATION

# THE SIMPSON METHOD



# HOW ABOUT MORE DIMENSIONS ?

# SIMPSON IN PHYSICS

In physics, 6D integral is the bread and butter

$$\int d\vec{r}\,d\vec{v}$$

# SIMPSON IN PHYSICS

Or even more…

$$\int \Pi_i d\vec{r_i}$$

# MONTE CARLO

# MONTE CARLO INTEGRATION

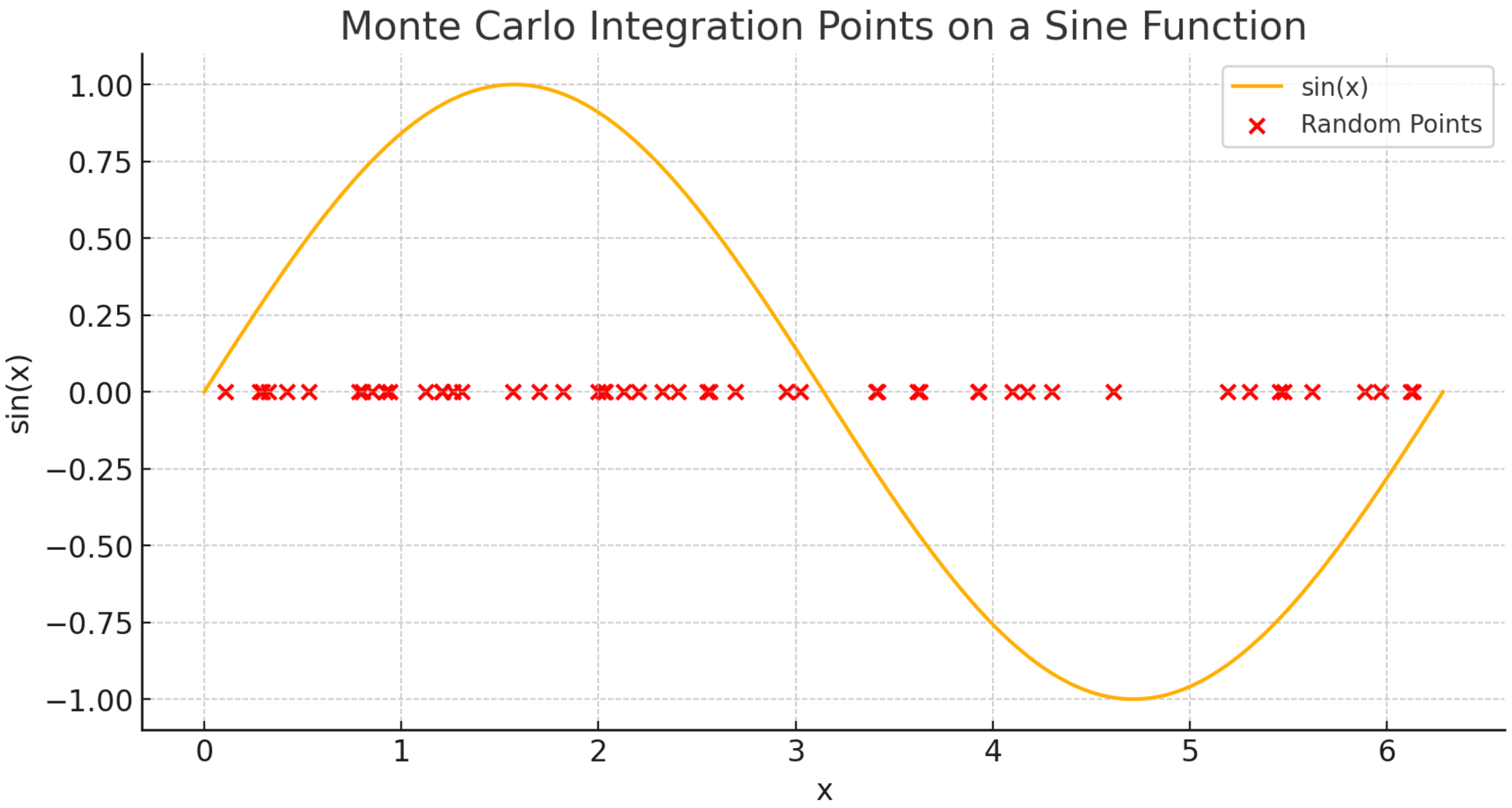
$$F = (b - a)\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

where $x_i \in [a, b]$

# MONTE CARLO INTEGRATION

$$F = (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

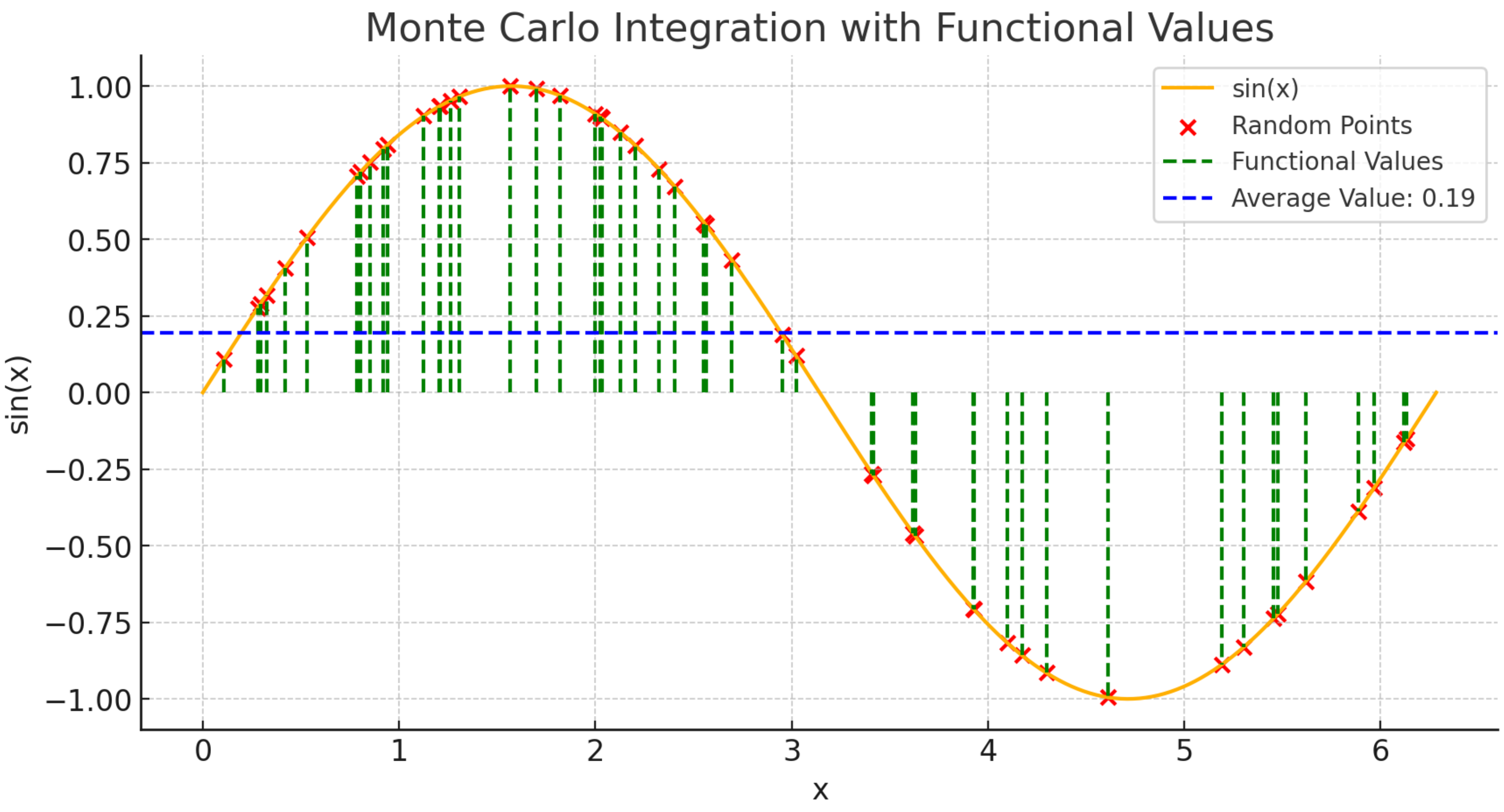
$$\text{where } x_i \in [a,b]$$



# MONTE CARLO INTEGRATION

$$F = (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

where $x_i \in [a, b]$



# COMPARISON

```python
def monte_carlo_sine_integral(n_samples):
    # Random samples across 6 dimensions
    points = np.random.rand(n_samples, 6)
    # Sum of points in each row to use in the sine function
    sum_points = np.sum(points, axis=1)
    # Compute sine
    values = np.sin(sum_points)
    # Average value times the volume of the unit cube
    return np.mean(values)


def simpson_sine_integral(n):
    # Simpson's rule coefficients
    h = 1 / n
    integral = 0
    for i1 in range(n + 1):
        for i2 in range(n + 1):
            for i3 in range(n + 1):
                for i4 in range(n + 1):
                    for i5 in range(n + 1):
                        for i6 in range(n + 1):
                            x1, x2, x3, x4, x5, x6 = i1 * h, i2 * h, i3 * h, i4 * h, i5 * h, i6 * h
                            weight = 1
                            if i1 == 0 or i1 == n:
                                weight *= 1/3
                            else:
                                weight *= 2/3 if i1 % 2 == 0 else 4/3
                            integral += np.sin(x1 + x2 + x3 + x4 + x5 + x6) * weight
    return integral * h**6
```

# COMPARISON

Converged value

```
Result: 0.109659
```

$10^6$ samples in both

```
Monte Carlo Result: 0.10905081983907904, Time: 0.20474004745483398 seconds
Simpson's Rule Result: 0.16927946314952982, Time: 4.333186864852905 seconds
```

# COMPARISON

Converged value

```
Result: 0.109659
```

$10^6$ samples in both

```
Monte Carlo Result: 0.10905081983907904, Time: 0.20474004745483398 seconds
Simpson's Rule Result: 0.16927946314952982, Time: 4.333186864852905 seconds
```

$6*10^7$ samples in Simpson

```
Simpson's Rule Result: 0.13703376827903233, Time: 182.18721294403076 seconds
```

$10^3$ samples in Monte Carlo

```
Monte Carlo Result: 0.09916473677114591, Time: 0.00025916099548339844 seconds
```

# COMPARISON

Converged value

```
Result: 0.109659
```

$10^6$ samples in both

```
Monte Carlo Result: 0.10905081983907904 ...nds
Simpson's Rule Result: 0.169279 ...nds
```

$6*10^7$ ...

```
Si... ...2.18721294403076 seconds
```

This depends on the dimensionality and the problem, but MC can be very efficient!

$10^3$ samples in Monte Carlo

```
Monte Carlo Result: 0.09916473677114591, Time: 0.00025916099548339844 seconds
```

# CAN IT GET EVEN BETTER ?

# IMPORTANCE SAMPLING

# IMPORTANCE SAMPLING

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \xrightarrow{\text{Law of large numbers}} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can also be a uniform distribution U(a,b)

$$\int_a^b f(x)$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can also be a uniform distribution U(a,b)

$$\int_a^b f(x) = \int_a^b f(x)\,\frac{b-a}{b-a}$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \xrightarrow{} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can also be a uniform distribution U(a,b)

$$\int_a^b f(x) = \int_a^b f(x)\frac{b-a}{b-a} = (b-a) * \int_a^b f(x)\frac{1}{b-a}$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

Function to integrate

U(a,b)

p(x) can also be a uniform distribution U(a,b)

$$\int_a^b f(x) = \int_a^b f(x)\frac{b-a}{b-a} = (b-a) * \int_a^b f(x)\frac{1}{b-a}$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can be anything, sometimes it cannot be sampled!

$$\int_a^b f(x)\,p(x) = \int_a^b f(x)\,\frac{p(x)}{q(x)}\,q(x)$$

# IMPORTANCE SAMPLING

Law of large numbers

$$\int_a^b f(x)\,p(x) = E_p[f(x)] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can be anything, sometimes it cannot be sampled!

$$\int_a^b f(x)\,p(x) = \int_a^b f(x)\frac{p(x)}{q(x)}q(x) = E_q\left[f(x)\frac{p(x)}{q(x)}\right] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)\frac{p(x_i)}{q(x_i)}$$

# IMPORTANCE SAMPLING

Law of large numbers

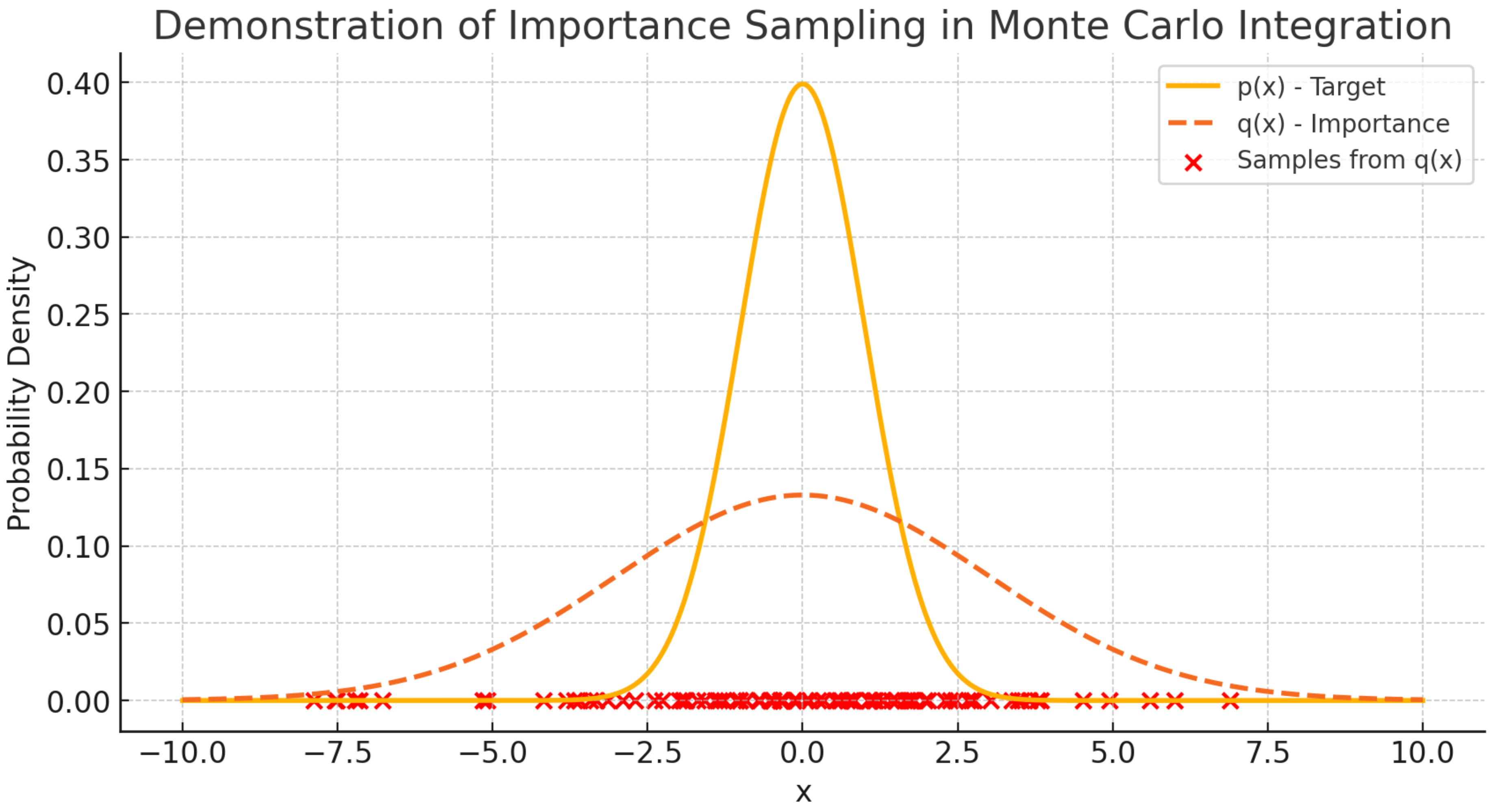
$$\int_a^b f(x)\,p(x) = E_p[f(x)] \xrightarrow{} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

p(x) can be anything, sometimes it cannot be sampled!

$$\int_a^b f(x)\,p(x) = \int_a^b f(x)\,\frac{p(x)}{q(x)}\,q(x) = E_q\left[f(x)\,\frac{p(x)}{q(x)}\right] \longrightarrow \frac{1}{n}\sum_{i=1}^{n} f(x_i)\,\frac{p(x_i)}{q(x_i)}$$

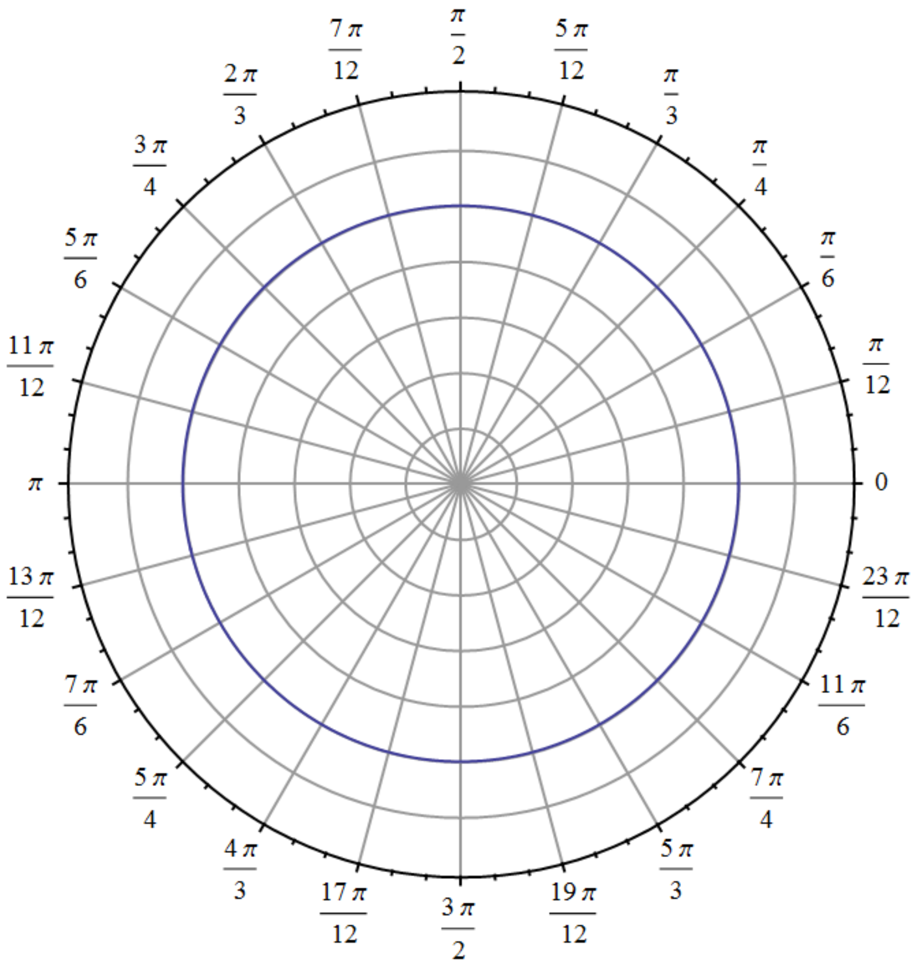
q(x) = 0 => f(x)p(x) = 0

# EXAMPLE

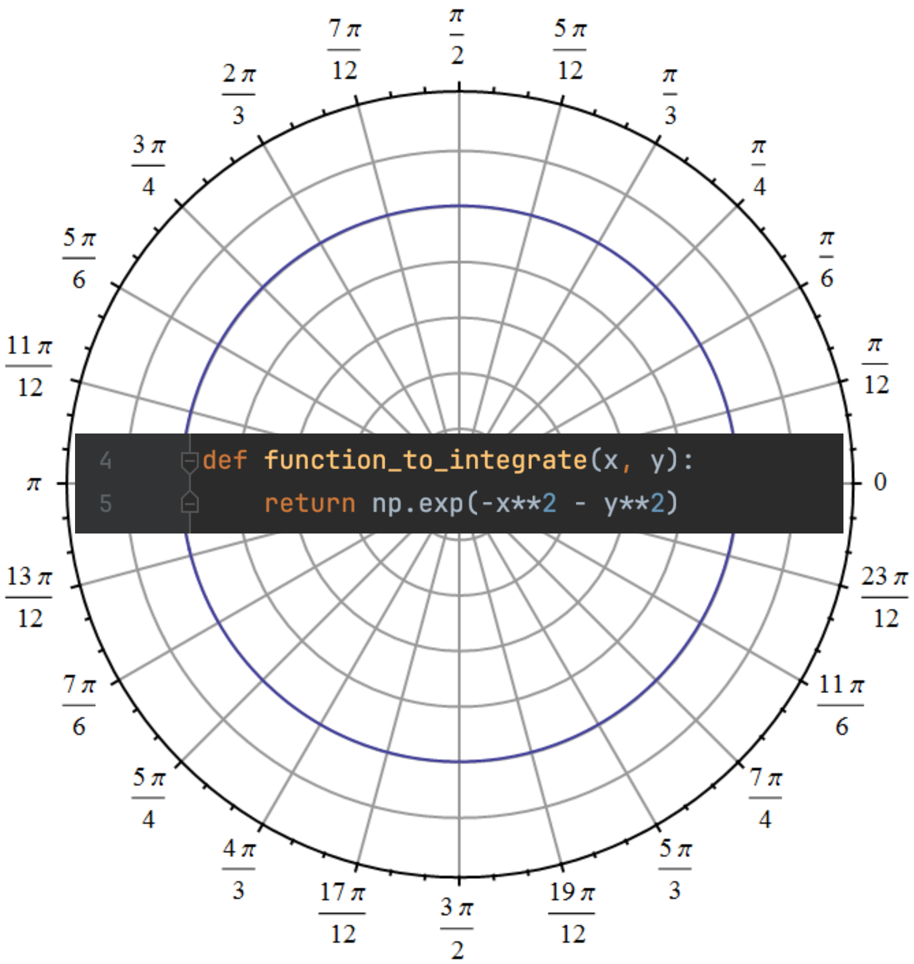


Demonstration of Importance Sampling in Monte Carlo Integration
p(x) - Target
q(x) - Importance
Samples from q(x)
Probability Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

# EXAMPLE FOR POLAR AND LOG-SCALE



# EXAMPLE FOR POLAR AND LOG-SCALE




```python
def function_to_integrate(x, y):
    return np.exp(-x**2 - y**2)
```

# EXAMPLE FOR POLAR AND LOG-SCALE

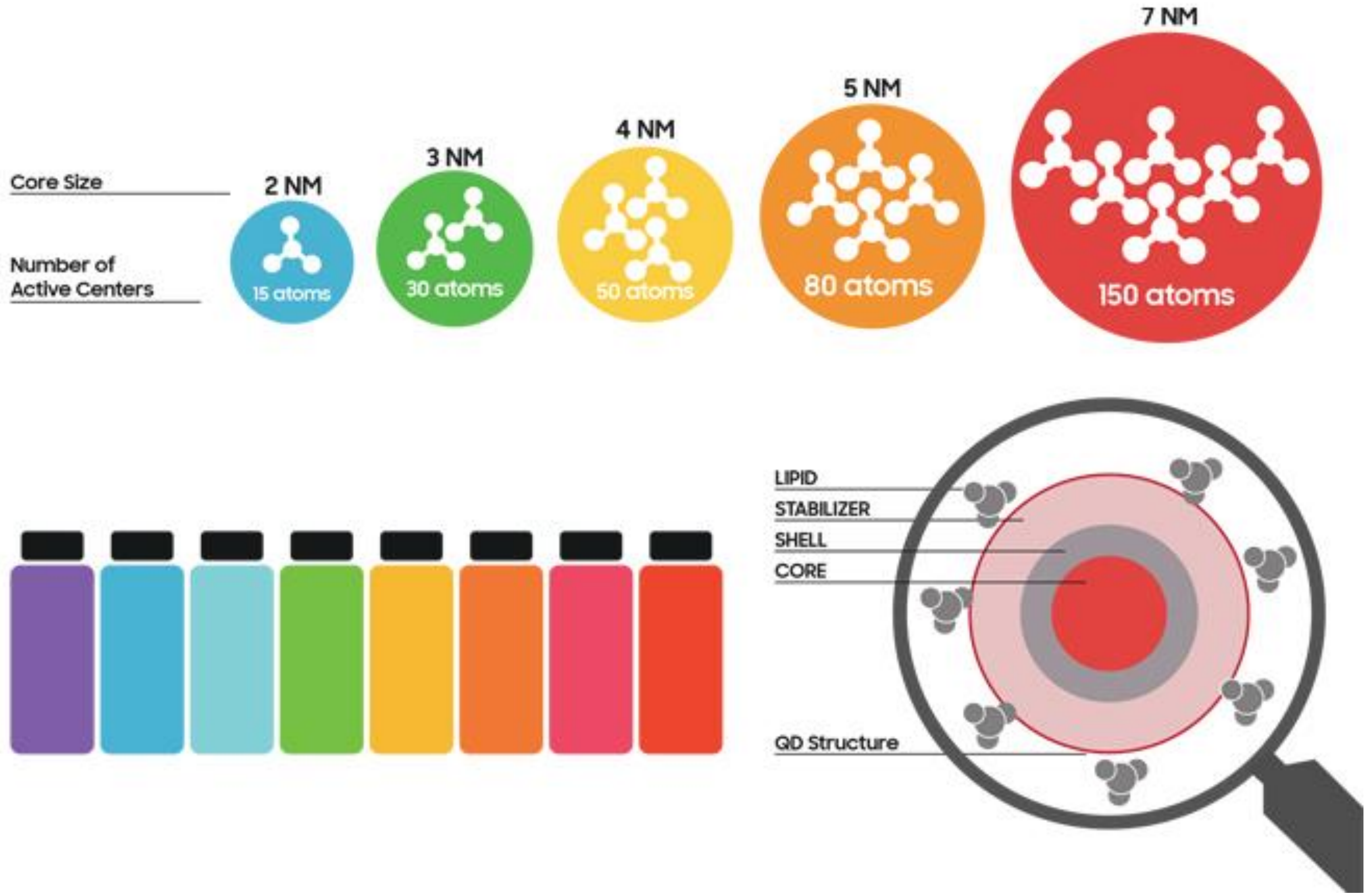
```python
def polar_MC(polar):
    size = 100000
    integral = 0.
    integration_radius = 4.
    if polar:
        for _ in range(size):
            r = np.random.random()*integration_radius
            phi = np.random.random()*2.*np.pi
            x = r*np.cos(phi)
            y = r*np.sin(phi)
            integral += function_to_integrate(x, y) * r
        integral = integral * 2.*np.pi * integration_radius / size
    else:
        for _ in range(size):
            length = 2. * integration_radius
            x = np.random.random()*length - length/2.
            y = np.random.random()*length - length/2.
            integral += function_to_integrate(x, y)
        integral = integral * length**2 / size
    print('POLAR: True integral should be pi ', '; MC:', integral, polar)
```

# EXAMPLE FOR POLAR AND LOG-SCALE

```python
def log_MC(log):
    size = 100000
    integral = 0.
    if log:
        for _ in range(size):
            x = np.random.uniform(-2, 7.)
            jacobian_MC_log = (10**x * np.log(10))*9.
            integral += 10**x * jacobian_MC_log
        integral = integral / size
    else:
        for _ in range(size):
            x = np.random.uniform(10**-2, 10**7)
            integral += x
        integral = integral*10**7 / size
    print('LOG: True integral should be 0.5*10**7*10**7 = 5*10**13; MC:', integral/10**13, '* 10**13', log)
```

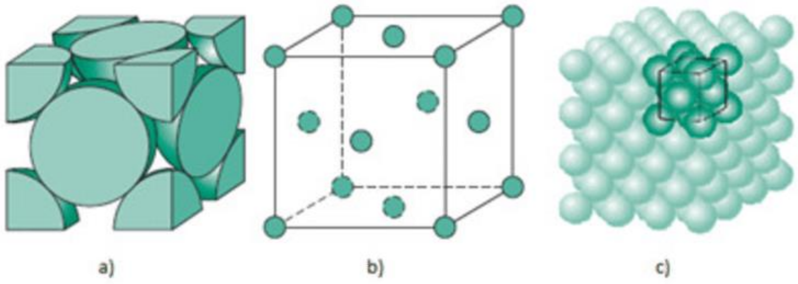
# THE CHALLENGE

# QUANTUM DOTS AS A NEW PARTICLE DETECTOR

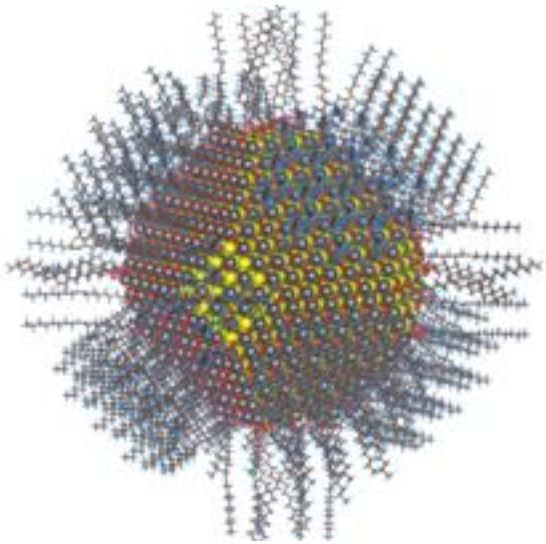



Fakulta jaderná a fyzikálně inženýrská

# QUANTUM DOTS AS A NEW PARTICLE DETECTOR




vs



# FINDING A SUM

```fortran
SUBROUTINE which_sums(ik1, ik2, alligk, tolerance, gsumindex)

  USE kinds,                            ONLY: DP
  USE wvfct,                            ONLY: npwx
  USE klist,                            ONLY: nks
  USE gvect,                            ONLY: g
  USE cell_base,                        ONLY: tpiba

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: ik1, ik2                         ! k-vector indices
  INTEGER, INTENT(IN) :: alligk(npwx, nks)
  REAL(DP), INTENT(IN) :: tolerance                       ! Tolerance for G-vector distance in RAU
  INTEGER, INTENT(OUT) :: gsumindex(npwx, npwx)           ! Index table for G-vector summation

  REAL(DP), DIMENSION(3) :: g1, g2, gaux                  ! G-vector coordinates
  INTEGER :: ig1, ig2, igaux                              ! G-vector indices
  REAL(DP) :: distance


  ! 0 is an exit value: should be discarded
  gsumindex(:,:) = 0


  DO ig1=1, npwx
     IF (alligk(ig1, ik1) == 0) CYCLE
     g1(:) = g(:, alligk(ig1, ik1))

     DO ig2=1, npwx
        IF (alligk(ig2, ik2) == 0) CYCLE
        g2(:) = g(:, alligk(ig2, ik2))

        DO igaux=1, npwx
           IF(alligk(igaux,ik2) == 0) CYCLE

           gaux(:) = g(:, alligk(igaux, ik2))

           gaux(:) = g1(:) + g2(:) - gaux(:)
           distance = tpiba * sqrt(sum(gaux(:)**2))

           IF (distance < tolerance) THEN
              gsumindex(ig2,ig1) = igaux
            ENDIF

           ENDDO
        ENDDO
     ENDDO


END SUBROUTINE which_sums
```

Fakulta jaderná
a fyzikálně inženýrská

# FINDING A SUM

For $G_1$ in G's

For $G_2$ in G's

For G in G's

```
SUBROUTINE which_sums(ik1, ik2, alligk, tolerance, gsumindex)

  USE kinds,                          ONLY: DP
  USE wvfct,                          ONLY: npwx
  USE klist,                          ONLY: nks
  USE gvect,                          ONLY: g
  USE cell_base,                      ONLY: tpiba

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: ik1, ik2                         ! k-vector indices
  INTEGER, INTENT(IN) :: alligk(npwx, nks)
  REAL(DP), INTENT(IN) :: tolerance                       ! Tolerance for G-vector distance in RAU
  INTEGER, INTENT(OUT) :: gsumindex(npwx, npwx)           ! Index table for G-vector summation

  REAL(DP), DIMENSION(3) :: g1, g2, gaux                  ! G-vector coordinates
  INTEGER :: ig1, ig2, igaux                              ! G-vector indices
  REAL(DP) :: distance


  ! 0 is an exit value: should be discarded
  gsumindex(:,:) = 0


  DO ig1=1, npwx
     IF (alligk(ig1, ik1) == 0) CYCLE
     g1(:) = g(:, alligk(ig1, ik1))

     DO ig2=1, npwx
        IF (alligk(ig2, ik2) == 0) CYCLE
        g2(:) = g(:, alligk(ig2, ik2))

        DO igaux=1, npwx
           IF(alligk(igaux,ik2) == 0) CYCLE

           gaux(:) = g(:, alligk(igaux, ik2))

           gaux(:) = g1(:) + g2(:) - gaux(:)
           distance = tpiba * sqrt(sum(gaux(:)**2))

           IF (distance < tolerance) THEN
              gsumindex(ig2,ig1) = igaux
             ENDIF

           ENDDO
        ENDDO
     ENDDO


END SUBROUTINE which_sums
```

Fakulta jaderná a fyzikálně inženýrská

# FINDING A SUM

1: 0 0 0
2: -1 0 0
3: 0 -1 0
4: 0 0 -1
.
.
.
21: -1 -1 1
22: -1 1 -1
23: -1 1 1
24: 1 -1 -1
25: 1 -1 1
26: 1 1 -1
27: 1 1 1
28: -2 0 0
29: 0 -2 0
30: 0 0 -2
31: 0 0 2
32: 0 2 0
33: 2 0 0
34: -2 -1 0
35: -2 0 -1
36: -2 0 1

```fortran
SUBROUTINE which_sums(ik1, ik2, alligk, tolerance, gsumindex)

  USE kinds,                          ONLY: DP
  USE wvfct,                          ONLY: npwx
  USE klist,                          ONLY: nks
  USE gvect,                          ONLY: g
  USE cell_base,                      ONLY: tpiba

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: ik1, ik2                        ! k-vector indices
  INTEGER, INTENT(IN) :: alligk(npwx, nks)
  REAL(DP), INTENT(IN) :: tolerance                      ! Tolerance for G-vector distance in RAU
  INTEGER, INTENT(OUT) :: gsumindex(npwx, npwx)          ! Index table for G-vector summation

  REAL(DP), DIMENSION(3) :: g1, g2, gaux                 ! G-vector coordinates
  INTEGER :: ig1, ig2, igaux                             ! G-vector indices
  REAL(DP) :: distance


  ! 0 is an exit value: should be discarded
  gsumindex(:,:) = 0


  DO ig1=1, npwx
     IF (alligk(ig1, ik1) == 0) CYCLE
     g1(:) = g(:, alligk(ig1, ik1))

     DO ig2=1, npwx
        IF (alligk(ig2, ik2) == 0) CYCLE
        g2(:) = g(:, alligk(ig2, ik2))

        DO igaux=1, npwx
           IF(alligk(igaux,ik2) == 0) CYCLE

           gaux(:) = g(:, alligk(igaux, ik2))

           gaux(:) = g1(:) + g2(:) - gaux(:)
           distance = tpiba * sqrt(sum(gaux(:)**2))

           IF (distance < tolerance) THEN
              gsumindex(ig2,ig1) = igaux
            ENDIF

           ENDDO
        ENDDO
     ENDDO


END SUBROUTINE which_sums
```

# FINDING A SUM

For $G_1$ in G's

For $G_2$ in G's

For $G_3$ in G's

If $G_1 + G_2 = G_3$, then

Store the index of the sum
$result(idx_1, idx_2) = idx_3$

```fortran
SUBROUTINE which_sums(ik1, ik2, alligk, tolerance, gsumindex)

  USE kinds,                          ONLY: DP
  USE wvfct,                          ONLY: npwx
  USE klist,                          ONLY: nks
  USE gvect,                          ONLY: g
  USE cell_base,                      ONLY: tpiba

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: ik1, ik2                        ! k-vector indices
  INTEGER, INTENT(IN) :: alligk(npwx, nks)
  REAL(DP), INTENT(IN) :: tolerance                      ! Tolerance for G-vector distance in RAU
  INTEGER, INTENT(OUT) :: gsumindex(npwx, npwx)          ! Index table for G-vector summation

  REAL(DP), DIMENSION(3) :: g1, g2, gaux                 ! G-vector coordinates
  INTEGER :: ig1, ig2, igaux                             ! G-vector indices
  REAL(DP) :: distance


  ! 0 is an exit value: should be discarded
  gsumindex(:,:) = 0


  DO ig1=1, npwx
    IF (alligk(ig1, ik1) == 0) CYCLE
    g1(:) = g(:, alligk(ig1, ik1))

    DO ig2=1, npwx
      IF (alligk(ig2, ik2) == 0) CYCLE
      g2(:) = g(:, alligk(ig2, ik2))

      DO igaux=1, npwx
        IF(alligk(igaux,ik2) == 0) CYCLE

        gaux(:) = g(:, alligk(igaux, ik2))

        gaux(:) = g1(:) + g2(:) - gaux(:)
        distance = tpiba * sqrt(sum(gaux(:)**2))

        IF (distance < tolerance) THEN
          gsumindex(ig2,ig1) = igaux
        ENDIF

      ENDDO
    ENDDO
  ENDDO


END SUBROUTINE which_sums
```

# FINDING A SUM

For $G_1$ in G's

For $G_2$ in G's

~~For $G_3$ in G's~~

~~If $G_1 + G_2 = G_3$, then~~

~~Store the index of the sum result($idx_1$,$idx_2$) = $idx_3$~~

```fortran
SUBROUTINE which_sums(ik1, ik2, allig k, tolerance, gsumindex)

  USE kinds,                          ONLY: DP
  USE wvfct,                          ONLY: npwx
  USE klist,                          ONLY: nks
  USE gvect,                          ONLY: g
  USE cell_base,                      ONLY: tpiba

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: ik1, ik2                          ! k-vector indices
  INTEGER, INTENT(IN) :: alligk(npwx, nks)
  REAL(DP), INTENT(IN) :: tolerance                        ! Tolerance for G-vector distance in RAU
  INTEGER, INTENT(OUT) :: gsumindex(npwx, npwx)            ! Index table for G-vector summation

  REAL(DP), DIMENSION(3) :: g1, g2, gaux                   ! G-vector coordinates
  INTEGER :: ig1, ig2, igaux                               ! G-vector indices
  REAL(DP) :: distance


  ! 0 is an exit value: should be discarded
  gsumindex(:,:) = 0


  DO ig1=1, npwx
    IF (alligk(ig1, ik1) == 0) CYCLE
    g1(:) = g(:, alligk(ig1, ik1))

    DO ig2=1, npwx
      IF (alligk(ig2, ik2) == 0) CYCLE
      g2(:) = g(:, alligk(ig2, ik2))

      DO igaux=1, npwx
        IF(alligk(igaux,ik2) == 0) CYCLE

        gaux(:) = g(:, alligk(igaux, ik2))

        gaux(:) = g1(:) + g2(:) - gaux(:)
        distance = tpiba * sqrt(sum(gaux(:)**2))

        IF (distance < tolerance) THEN
          gsumindex(ig2,ig1) = igaux
         ENDIF

        ENDDO
      ENDDO
    ENDDO


END SUBROUTINE which_sums
```


LET THE HUNGER GAMES
BEGIN
memegenerator.net


That's all Folks!